

Taller en Sala 2 Recursión



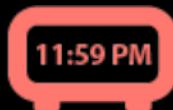
Objetivo: Identificar el caso base y el caso general de un problema definido recursivamente



Consideraciones: Lean y verifiquen las consideraciones de entrega,



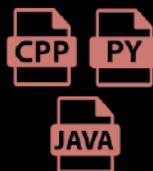
Trabajo en
Parejas



Mañana,
plazo de
entrega



Docente entrega
plantilla de
código en
GitHub



Sí .cpp, .py
o .java



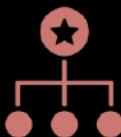
No .zip, .txt,
html o .doc



Alumnos
entregan
código sin
comprimir
GitHub



En la carpeta Github del curso, hay un código iniciado y un código de pruebas (tests) que pueden explorar para solucionar los ejercicios



Estructura del documento: a) Datos de *vida real*, b) Introducción a un problema, c) Problema a resolver, d) Ayudas. Identifiquen esos elementos así:

a)



b)



c)



d)



Ejercicios a resolver



El máximo común divisor se puede usar para determinar el tamaño de las baldosas para el piso en videojuegos como *The Sims 3* de tal forma que se ajusten a las dimensiones del piso sin dejar sobrantes

- 1** En el videojuego *The Sims 3*, es posible elegir el tipo de baldosas que uno desea para el piso de su casa. Algo curioso del juego es que las baldosas se ajustan exactamente al tamaño de la habitación, sin dejar sobrantes. Para lograr esto, se debe elegir un tamaño de baldosa cuadrada que divida exactamente tanto el largo como el alto de la habitación.



Implementen el algoritmo de Euclides de forma recursiva.



El problema de la suma de subconjuntos fue propuesto en la maratón de Dropbox del 2011, vean <https://bit.ly/1SiAzIk>

ESTRUCTURA DE DATOS 1

Código ST0245

2

En el videojuego *Dark Souls 2*, cada jugador tiene un inventario de elementos que puede cargar. Los elementos que puede cargar no deben superar un peso máximo definido en el juego. Ejemplos de estos elementos son armas y accesorios.

Un problema que ocurre con frecuencia en el juego es que uno encuentra tesoros con muchas armas y accesorios, más de los que uno puede cargar. Una solución es escoger un subconjunto de estos elementos cuya suma sea igual al peso máximo.



Implementen un algoritmo recursivo que, dados los pesos de un grupo de elementos, determine si existe o no existe un subgrupo de elementos cuya suma sea igual a un peso máximo.

3

En el videojuego *Dark Souls 2*, para elegir los elementos que uno desea tener en su inventario, no siempre es tan trivial como encontrar un subgrupo de elementos cuya suma de pesos sea el peso máximo.

En muchas ocasiones se debe tener en cuenta que, ciertas características del personaje, dan mejor desempeño a ciertas armas o que las armas se pueden vender a otros jugadores. Una solución es dejar que el usuario sea quien tome la decisión y que el sistema simplemente le muestre cuáles son los posibles subgrupos.

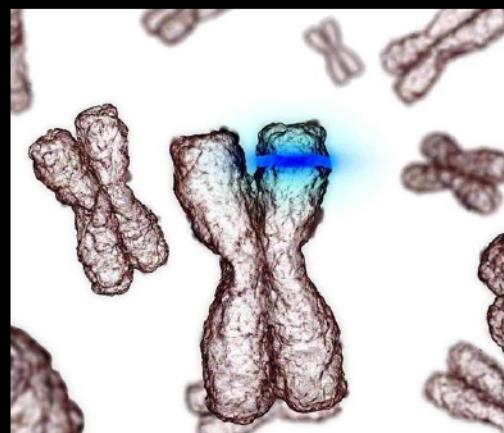


Implementen un algoritmo recursivo que, dado un grupo de elementos, le muestre, en pantalla, todos los posibles subgrupos que se pueden formar con esos elementos.

4

[Ejercicio Opcional] El genoma humano fue decodificado en su totalidad en el 2003. Desde ese entonces, una de las grandes líneas de investigación en Bioinformática es encontrar, en el genoma humano, las secuencias de ADN responsables de la aparición de diferentes tipos de cáncer.

Las cadenas de ADN son, simplemente, cadenas de caracteres. El problema de encontrar un patrón cancerígeno en el ADN de una persona, en algunos casos, puede verse como el problema de la subsecuencia común más larga.



El problema de la subsecuencia común más larga es el siguiente: Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero no necesariamente de forma contigua.



Implementen un algoritmo que calcule la longitud de la subsecuencia común más larga a dos cadenas de caracteres.

Ayudas para resolver los Ejercicios

Ejercicio 1.....	<u>Pág. 5</u>
Ejercicio 2.....	<u>Pág. 6</u>
Ejercicio 3.....	<u>Pág. 9</u>



Ejercicio 1



Pista 1: El algoritmo de Euclides calcula el máximo común divisor de dos enteros (MCD). El MCD de dos números es el entero más grande que divide a ambos.



Ejemplo 1, El máximo común divisor de 102 y 68 es 34. El algoritmo de Euclides propone calcular el MCD de esta forma:

“El MCD de p y q es el mismo que el MCD de $p \% q$ y q”, donde % es la operación de módulo. Asuman que el usuario siempre entrega p y q de tal forma que $p > q$. **Implementen un método que calcule el algoritmo de Euclides.**



Pista 2: Definir la condición de parada.



Pista 3: Paso 1: Definir la firma de la función

```
public static int gcd(int p, int q) {  
}
```



Pista 4: En el documental de Algoritmos de la BBC, en <https://www.youtube.com/watch?v=Q9HjeFD62Uk>, en el minuto 6:19, se muestra gráficamente el funcionamiento de este algoritmo.:



Pista 5: Este simulador puede ayudarles a entender **el algoritmo de Euclides** <https://visualgo.net/recursion>



Error Común:



Ejercicio 2



Pista 1: Pepito escribió un algoritmo que, dado un arreglo de enteros, decide si es posible escoger un subconjunto de esos enteros, de tal forma que la suma de los elementos de ese subconjunto sea igual al parámetro **target**.

El parámetro **start** funciona como un contador y representa un índice en el arreglo de números **nums**. Ayúdenle a completarlo, por favor, si es tan amable.



Ejemplo 1, si preguntan si hay un subconjunto de [2,4,8] que sume 10, la respuesta es verdadero. **Como Ejemplo 2**, si preguntamos si hay un subconjunto de [2,4,8] que sume 20, la respuesta es falso.

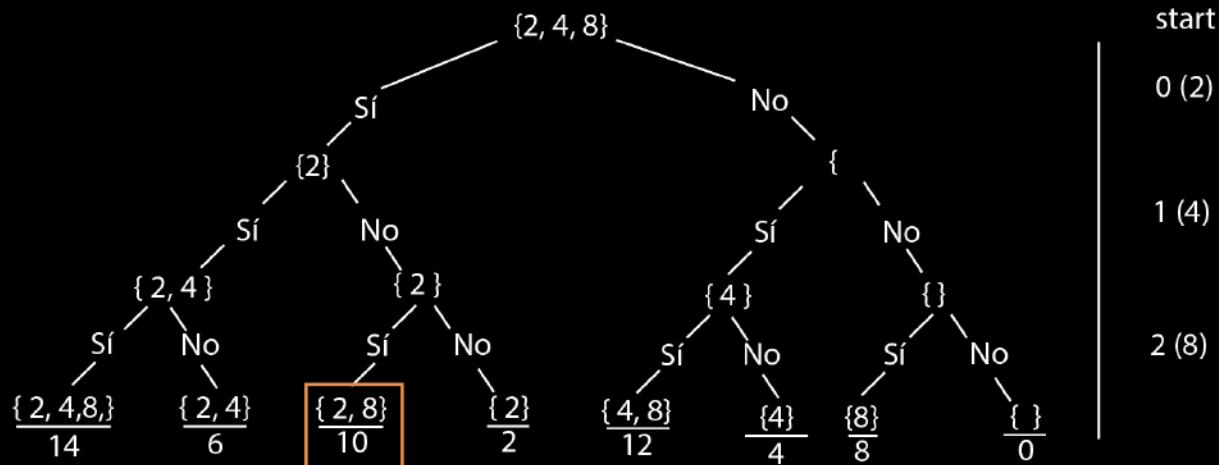
```
private static boolean sumaGrupo(int start, int[] nums, int target) {
    }

    public static boolean sumaGrupo(int[] nums, int target) {
        return sumaGrupo(0, nums, target);
    }
}
```

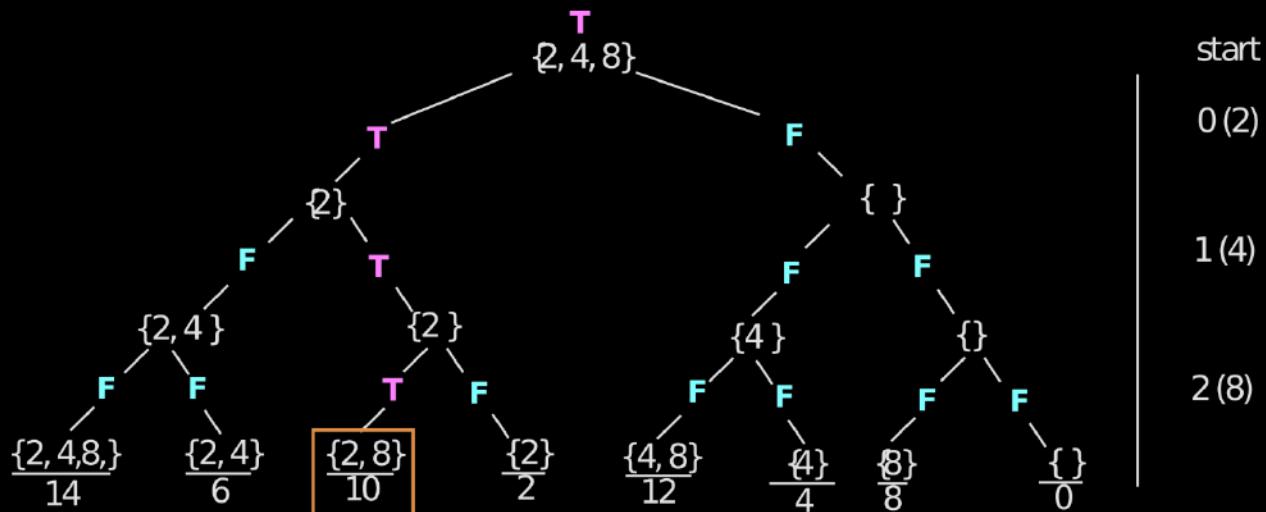
ESTRUCTURA DE DATOS 1
Código ST0245



Pista 2: Se deben hacer dos llamados recursivos. **Por ejemplo**, a continuación, se muestra el árbol de ejecución para encontrar si un subconjunto de $\{2,4,8\}$ suma 10.



Pista 3: El retorno de los 2 llamados recursivos se une en una sola respuesta utilizando el operador de disyunción (*or*).



 **Error Común 1:**



 **Error Común 2:** Este algoritmo falla porque al llamarse recursivamente con el parámetro `start` se queda en una recursión infinita.

```
public boolean sumaGrupo(int start, int[] nums, int target)
{
    if (start >= nums.length) return target == 0;
    return sumaGrupo(start+1, nums, target - nums[start])
        || sumaGrupo(start, nums, target );
}
```

 **Error Común 3:** Este algoritmo falla porque al llamarse recursivamente con el parámetro `start-1` se sale del arreglo cuando `start = 0`

ESTRUCTURA DE DATOS 1

Código ST0245

```
public boolean sumaGrupo(int start, int[] nums, int target) {
    if (start >= nums.length) return target == 0;
    return sumaGrupo(start+1, nums, target - nums[start])
        || sumaGrupo(start-1, nums, target );
}
```



Error Común 4: Este algoritmo falla porque al llamarlo recursivamente con el parámetro `start` se sale del arreglo cuando `start = 0`.

```
public boolean sumaGrupo(int start, int[] nums, int target) {
    if (start >= nums.length) return target == 0;
    return sumaGrupo(start+1, nums, target - nums[start])
        || sumaGrupo(start+1, nums, target - nums[start - 1] );
}
```



Ejercicio 3



Pista 1: Escriban un programa que calcule las combinaciones de letras de una cadena. Una combinación es un subconjunto de los n elementos, independiente del orden. Existen 2^n subconjuntos. Como un ejemplo, para “abc”, debe dar esta respuesta:

a ab abc ac b bc c

```
public static void combinations(String s) {
    combinationsAux("", s);
}
```



Pista 2: Calculen los subconjuntos, en una hoja de papel, para la cadena “abcd”



Pista 3: Calculen los subconjuntos, en una hoja de papel, para la cadena “abcd”



Pista 4: Si tienen dudas sobre cuáles son los subconjuntos de un conjunto, vean este video <https://www.youtube.com/watch?v=Bxv2KvlItxs>. No se preocupen por el orden en que obtienen las combinaciones

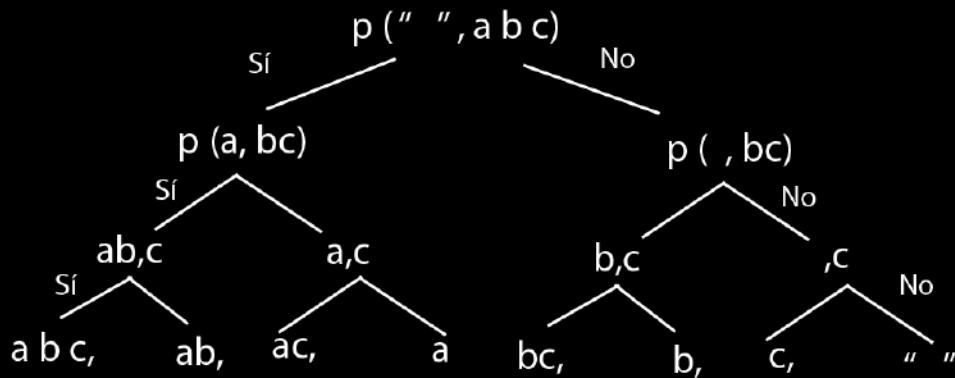


Pista 5: Primero defina una función auxiliar de esta forma:

```
private static void combinationsAux(String base, String s)
{
    ...
}
```

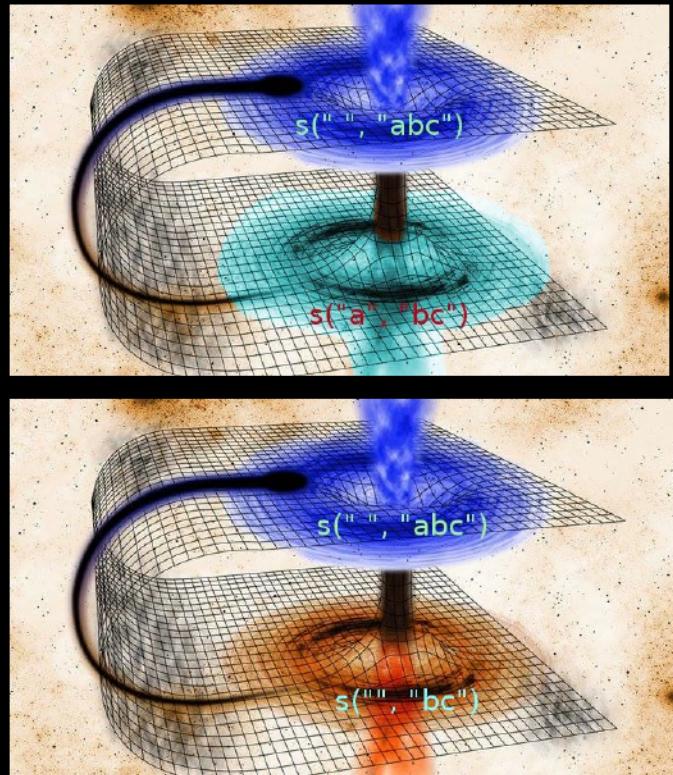


Pista 6: La función recursiva debe generar el siguiente árbol de ejecución para generar los subconjuntos de la cadena “abc”.





Pista 7: Suponga que un llamado recursivo es como crear un universo paralelo donde todo es igual excepto el valor de los parámetros



Error Común 1: Un error común es calcular los prefijos en lugar de los subconjuntos, como se muestra en el siguiente ejemplo. La solución es hacer 2 llamados recursivos y no uno solo.

```
public static void prefijos(String base, String s) {
    if (s.length() == 0){
        System.out.println(base);
    }
    else {
        prefijos (base + s.charAt(0), s.substring(1));
        System.out.println(base);
    }
}
```

PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

UNIVERSIDAD
EAFIT[®]

Acreditación
Institucional
Renovación
2018 - 2026
Resolución MEN 2158 de 2018



Ejercicio 4

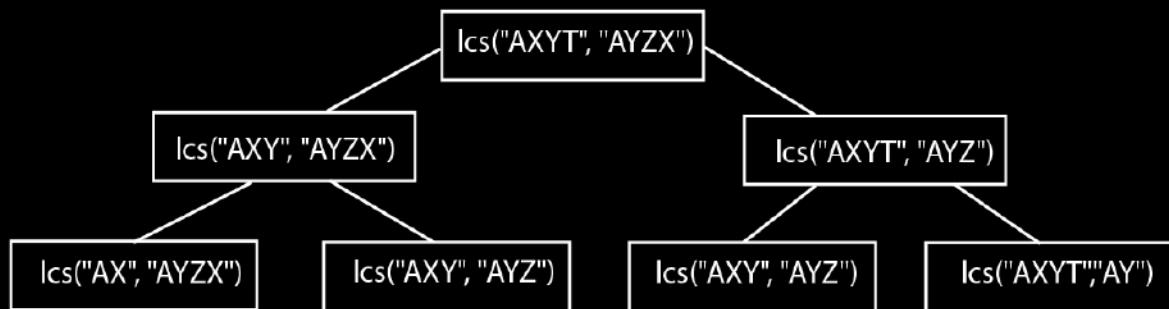


Ejemplo 1: “abc”, “abg”, “bdf”, “aeg” y “acefg” son subsecuencias de “abcdefg”. Entonces, para una cadena de longitud n existen 2^n posibles subsecuencias.



Ejemplo 2: Consideren los siguientes ejemplos para el problema:

Para “ABCDGH” y “AEDFHR” es “ADH” y su longitud es 3. Para “AGGTAB” y “GXTXAYB” es “GTAB” y su longitud es 4. Una forma de resolver este problema es usando *backtracking*, como un ejemplo, para las cadenas “AXYT” y “AYZX”, dada una función recursiva los que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:



Pista 2: Completén el siguiente método:

```

// Precondición: Ambas cadenas x, y son no vacías
public static int lcstdyn(String x, String y) {
    ...
}
  
```

¿Alguna inquietud?

CONTACTO

Docente Mauricio Toro Bermúdez
Teléfono: (+57) (4) 261 95 00 Ext. 9473
Correo: mtorobe@eafit.edu.co
Oficina: 19- 627

Agenden una cita dando clic en la pestaña
-Semana- de <http://bit.ly/2gzVg10>